Universidade de Aveiro
2007
Departamento de Electrónica, Telecomunicações e Informática

PEDRO MIGUEL DE JESUS FIGUEIREDO

REMOÇÃO DO EFEITO DE OLHOS VERMELHOS EM FOTOGRAFIA DIGITAL

Universidade de Aveiro
2007
Departamento de Electrónica, Telecomunicações e Informática

PEDRO MIGUEL DE JESUS FIGUEIREDO

REMOÇÃO DO EFEITO DE OLHOS VERMELHOS EM FOTOGRAFIA DIGITAL

dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica do Dr. Armando José Formoso de Pinho, Professor Associado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Dedico este trabalho à minha namorada e família.

o júri

presidente

Doutor Tomás António Mendes de Oliveira e Silva
Prof. Associado da Universidade de Aveiro

orientador

Doutor Armando José Formoso de Pinho
Prof. Associado da Universidade de Aveiro

Doutor Vitor Manuel Jesus Filipe
Prof. Auxiliar da Universidade de Trás-os-Montes e Alto Douro

agradecimentos

Agradeço a todos que directamente, ou indirectamente, contribuíram de algum modo para este trabalho. A todos os que me deram apoio quando precisei e me incentivaram quando mais precisava.

Em especial um muito obrigado ao engenheiro António J. R. Neves, sem o qual a conclusão deste trabalho não tinha sido possível.

palavras-chave

JPEG, efeito de olhos vermelhos, codificação sequencial, codificação, descodificação, algoritmo, Huffman, quantização, codificação por transformada, codificação diferencial.

resumo

Nesta dissertação apresentamos um sistema para remoção do efeito de olhos vermelhos em fotografia digital no formato JPEG. Este sistema é semi-automático e tem como principal característica apenas re-codificar os blocos de imagem correspondentes à zona modificada, deixando todos os outros blocos inalterados. Para a sua implementação foi necessário um estudo detalhado da norma JPEG de modo a desenvolver um descodificador para este tipo de imagens. Para re-codificar os blocos modificados foi desenvolvido um algoritmo simples que, usando a informação contida no ficheiro, codifica os blocos modificados de uma forma eficiente. Para a eliminação do efeito de olhos vermelhos foram desenvolvidos diversos algoritmos e efectuado um conjunto de testes de modo a comparar os resultados obtidos.

keywords

JPEG, red eye effect, sequential coding, encoding, decoding, algorithm, Huffman, quantization, transform coding, differential coding.

abstract

In this dissertation we present a system to remove the red eye effect of digital photos in JPEG format. This system is semi-automatic and its principal goal is that of only encoding the modified blocks of the image, leaving all the other blocks unchanged. For this implementation it was necessary a detailed study of the JPEG norm, in order to develop a decoder for this type of images. To re-encode the modified blocks it was developed a simple algorithm that, using the information contained in the file, encodes the modified blocks efficiently. To remove the red eye effect diverse algorithms had been developed and a set of tests were made in order to compare the results.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A fotografia digital encontra-se hoje bastante vulgarizada, permitindo a qualquer pessoa com um computador pessoal efectuar um sem fim de operações sobre as imagens adquiridas pela máquina fotográfica. Uma das funcionalidades frequentemente encontradas em programas para manipulação de imagem fotográfica consiste na remoção do tão desagradável efeito de olhos vermelhos.

Este efeito deve-se à iluminação do fundo do olho provocada pelo flash da máquina fotográfica. Embora muitas das máquinas possuam sistemas que tentam evitar este fenómeno, emitindo feixes de luz antes do flash para obrigar as pupilas dos olhos a fechar, o certo é que muitas vezes esses sistemas falham. Neste caso, a única possibilidade é aplicar um pós-processamento à imagem digital.

Actualmente, quase todas as máquinas fotográficas digitais armazenam as fotografias num formato comprimido, tipicamente usando a norma JPEG (Joint Pictures Expert Group. Em geral, quando se pretende efectuar algum pós-processamento numa imagem, ela é primeiro descomprimida, depois processada e, finalmente, re-comprimida. Como a compressão usada introduz perda de informação, este processo de descompressão/re-compressão introduz degradação adicional e/ou um aumento inaceitável do tamanho do ficheiro.

O formato JPEG permite o armazenamento de imagens com 24 bits de representação de cor. Isso significa que este formato aceita 16,8 milhões de cores. O JPEG é um dos formatos de imagens mais populares e isso deve-se à capacidade de formar imagens fiéis à original. Além disso, os ficheiros em JPEG costumam ter tamanhos bastante mais reduzidos que os originais. Na verdade, uma imagem JPEG pode ter o seu tamanho cerca de 100 vezes inferior

ao original e mesmo assim conseguir ter uma qualidade aceitável.

O JPEG utiliza um algoritmo de compactação que aproveita limitações do olho humano. No entanto, mesmo sabendo-se que os ficheiros JPEG podem trabalhar com até 16,8 milhões de cores, o olho humano não é capaz de distinguir todas essas cores. Assim, é possível retirar uma série de informações que representam cores em imagens e manter apenas aquelas visíveis ao olho humano. Por outras palavras, o formato JPEG retira da imagem aquilo que os humanos não conseguem ver. Esse processo é conhecido como compressão perceptual. Isso faz com que imagens bastante realistas sejam criadas, ao mesmo tempo que os ficheiros ficam bastante mais pequenos.

Um aspecto importante no JPEG é o facto de os ficheiros poderem ter diferentes níveis de compressão. Quanto maior for o nível de compressão (mais informação é retirada do ficheiro), menor será o tamanho do ficheiro, contudo a sua qualidade será inferior.

Uma desvantagem do JPEG é que a imagem normalmente perde qualidade cada vez que é re-comprimida. É esse o principal foco deste trabalho. Tendo em conta esse problema, tentamos minimizá-lo o quanto possível ao modificar apenas a parte da foto que é alterada ao remover o efeito de olho vermelho e manter todo o resto da fotografia inalterada para que a perda de qualidade seja minimizada e restringida a uma área limitada e controlada.

1.1 O porquê dos olhos vermelhos

O efeito dos olhos vermelhos, em fotografia, consiste no surgir de pontos vermelhos nos olhos das pessoas e animais retratados em fotografias, resultantes do reflexo da luz do flash.

Em algumas espécies de animais, existe uma camada reflectora atrás da retina que melhora a visão nocturna e que aumenta este efeito, o que conduz por vezes a variações da cor da luz reflectida de espécie para espécie. Os olhos dos gatos, por exemplo, podem reflectir em fotografias de flash, a cor azul, amarela, rosa, ou verde.

Figura 1.1: Gato com efeito de olhos amarelos

Para o reflexo atingir o sensor da câmara (ou o filme, pois as convencionais também sofrem com o problema), é preciso que o ângulo de incidência do flash seja muito pequeno, o que acontece principalmente em duas situações:

- Fotos com flash de pessoas distantes - é muito raro acontecer em câmaras compactas pois o flash nestas tem um alcance muito reduzido. Quanto mais perto do motivo a câmara estiver, menor a probabilidade de olhos vermelhos;

- Flash muito próximo da lente - é a principal explicação para o número crescente de fotos com olhos vermelhos, uma vez que com câmaras cada vez menores, o flash quase sempre fica perto demais da objectiva.

Alguns modelos tentam reduzir o problema com o flash que se abre acima do corpo da câmara, mais afastado da lente. A maioria, no entanto, conta apenas com o tal modo de redução de olhos vermelhos. O que ele faz é emitir alguns rápidos flashes de luz antes do flash propriamente dito com o intuito de fazer com que as pupilas das pessoas se contraiam, diminuindo assim o orifício pelo qual a luz teria que entrar e ser reflectida de volta para a câmara. Na prática, ajuda muito pouco.

Uma outra medida a adoptar no sentido de evitar o aparecimento de olhos vermelhos é a de acender as luzes do ambiente antes de fotografar, mesmo quando o flash for capaz de iluminar a cena adequadamente, isto porque é muito mais provável encontrar olhos vermelhos em fotos tiradas no escuro, onde as pessoas estavam com as pupilas totalmente dilatadas, do que em ambientes bem iluminados.

1.2 Objectivos

Implementar um sistema que permita a remoção do efeito de olhos vermelhos em fotografias digitais em formato JPEG. Este sistema deverá ser automático ou semi-automático, sendo caracterizado por só re-codificar os blocos de imagem correspondentes às zonas alteradas, deixando todos os outros blocos inalterados.

Em termos gerais, o sistema é constituído pelas seguintes fases:

- Numa primeira fase é necessário descodificar o ficheiro JPEG de modo a que depois seja possível visualizar/editar a imagem.
- De seguida é necessário que, de forma automática, ou semi-automática, se elimine o efeito de olhos vermelhos.
- Depois é necessário re-codificar somente os blocos alterados, de modo a impedir uma degradação, ou aumento do tamanho, do ficheiro.

Na Figura 1.2 está esquematizado o objectivo do trabalho:

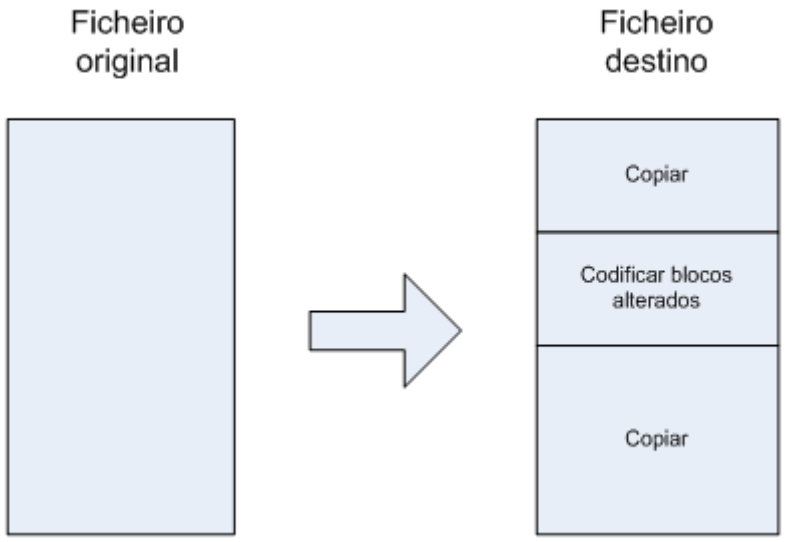

Figura 1.2: Objectivos do trabalho

O trabalho foi realizado em linguagem C, no sistema operativo Linux.

Capítulo 2

Norma JPEG

2.1 Descrição

JPEG é um acrónimo de Joint Photographic Experts Group que é o nome original do comité, fundado em 1986, que criou uma norma com o mesmo nome, JPEG, em 1990. Essa norma é a junção de esforços de duas das maiores organizações de normas do mundo: ISO/IEC (International Organization for Sandardization / International Electrotechnical Commission) e o ITU-T (International Telecommunication Union - Telecommunication Standardization Sector).

O JPEG é talvez um dos formatos de compressão de imagens mais utilizados hoje em dia, especialmente utilizado para comprimir imagens fotográficas. É um formato de ficheiro de imagem muito popular, que permite uma grande compressão mantendo uma boa qualidade.

A norma JPEG foi desenvolvida para comprimir imagens paradas, em tons contínuos de cinza ou de cor, de cenas do mundo real e imagens naturais, não tendo um bom desempenho em imagens que apresentem descontinuidades nas cores ou nos tons de cinza. Outro detalhe importante é que o JPEG no modo mais comummente utilizado leva a uma compressão com perdas na imagem. É de realçar que estas perdas são proporcionais ao factor de compressão desejado.

A norma JPEG é razoavelmente complexa porque, mais que definir um formato de arquivo de imagem, ela define um grande número de relacionamentos entre técnicas de compressão de imagens. Na norma JPEG existem quatro modos de compressão dos dados. São eles o modo sequencial, o modo progressivo, o modo hierárquico e o modo sem perdas [2].

2.1.1 JPEG progressivo

No JPEG progressivo, as componentes das imagens são codificadas em múltiplas passagens. A compressão de dados de cada componente é feita no mínimo em duas passagens. A passagem inicial cria uma versão com pouca definição da imagem, enquanto passagens subsequentes fazem o seu refinamento. Imagens progressivas vão sendo visualizadas conforme são descodificadas, dando ao utilizador uma ideia do conteúdo da imagem após uma pequena quantidade de dados ter sido transmitida. Mais difícil de ser implementado, é mais recomendado quando a velocidade de processamento é relativamente mais rápida, excedendo a velocidade de transmissão da imagem através de uma rede.

Figura 2.1: Exemplo de uma imagem progressiva

2.1.2 JPEG hierárquico

O JPEG hierárquico é tido como um modo super-progressivo em que a imagem é subdividida num certo número de sub-imagens chamadas frames (colecção de uma ou mais passagens). No modo hierárquico, o primeiro frame cria uma versão de baixa resolução da imagem. Os frames restantes refinam a imagem por incremento da resolução. Os principais obstáculos do modo hierárquico são: a complexidade de sua implementação, mais processamento, maior quantidade de dados a serem transmitidos e, portanto, menor taxa de compressão.

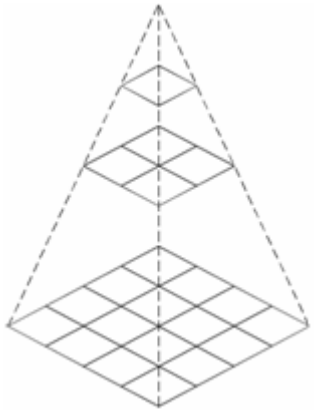

Figura 2.2: Exemplo de uma imagem hierárquica

2.1.3 JPEG sem perdas

A norma JPEG permite um modo de compressão sem perdas. Foi escolhido para o efeito um método preditivo. Neste modo a codificação é feita com o auxílio de um preditor e de codificação estatística. O preditor estima o valor do pixel actual (X) com base na informação do valor dos pixels vizinhos (A, B e C) [5] [12].

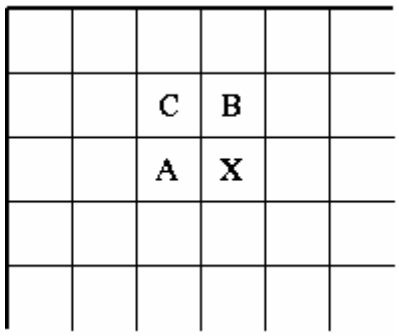

Figura 2.3: Pixels vizinhos (A, B e C) usados na predição do valor do pixel que se quer codificar (X)

A norma JPEG contempla a utilização de 7 preditores diferentes:

Predictor	**Prediction**
1	A
2	B
3	C
4	A + B - C
5	A + (B - C)/2
6	B + (A - C)/2
7	(A + B)/2

Tabela 2.1: Preditores

O valor a codificar é a diferença entre o valor estimado pelo preditor e o valor do pixel a codificar.

O JPEG sem perdas preserva de forma exacta a imagem original apresentando baixa taxa de compressão. Na maioria dos casos não é competitivo.

2.1.4 JPEG sequencial

No JPEG sequencial, cada componente de cor é completamente codificada apenas numa passagem, ou seja, um bloco de dados é comprimido de uma só vez, directamente da imagem, para uma ou mais componentes.

Figura 2.4: Exemplo de uma imagem sequencial

Normalmente, quando há referência à norma JPEG trata-se do modo sequencial, cujo grau de qualidade e velocidade de descompressão podem ser variados de acordo com os parâmetros da compressão. Isso significa que o tamanho do arquivo da imagem em processamento pode ser regulado de acordo com a qualidade final da imagem desejada.

Tipicamente, para imagens coloridas, a norma JPEG obtém taxas de compressão entre 10:1 a 20:1 sem perdas visíveis, taxas de 30:1 a 50:1 com pequenos defeitos na imagem, enquanto que, para uma qualidade muito pobre da imagem, pode ser obtida uma taxa de compressão de até 100:1. Para imagens em tons de cinza, não se obtêm taxas tão altas de compressão devido ao facto de os olhos humanos serem mais sensíveis a variações espaciais das intensidades luminosas do que das cores; com isso o limite da taxa de compressão sem perdas visíveis fica em torno de 5:1. Num processo repetido de compressão e descompressão as perdas na imagem são acumulativas.

A norma JPEG é específica para imagens. Contudo, frequentemente ouvem-se referências ao motion JPEG ou MJPEG para vídeo. Vários fabricantes e vendedores de equipamentos médicos têm aplicado JPEG aos frames de uma sequência de vídeo chamando o resultado de MJPEG [13].

Para vídeo, a norma de compressão/descompressão mais utilizada é o MPEG (Motion Picture Experts Group), a qual utiliza muitas das técnicas do JPEG, além de explorar a redundância inter-frames, geralmente presente numa sequência de vídeo devido ao facto da cena não variar muito entre frames sucessivas, elevando a taxa de compressão cerca de três vezes em relação ao método MJPEG para semelhante qualidade da imagem. Contudo, tem o inconveniente de requerer um buffer que permita armazenar várias frames, enquanto que

a norma JPEG não necessita de nenhum buffer para frames, pois processa cada frame independentemente [13] [14].

Outra vantagem da norma JPEG é apresentar uma grande simetria, em termos de complexidade, entre os codificadores e descodificadores. Por esta razão, em algumas aplicações, onde é necessário/possível processar vídeo mas em que há limites na capacidade de processamento, utiliza-se normalmente a norma JPEG (MJPEG), mesmo perdendo em taxa de compressão.

2.1.5 Princípios da codificação JPEG no modo sequencial

A norma JPEG pode ser descrita através dos blocos da Figura 2.5, a qual pode ser expandida para a Figura 2.6.

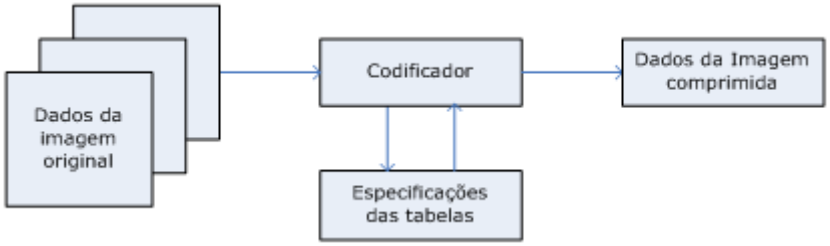

Figura 2.5: Codificação

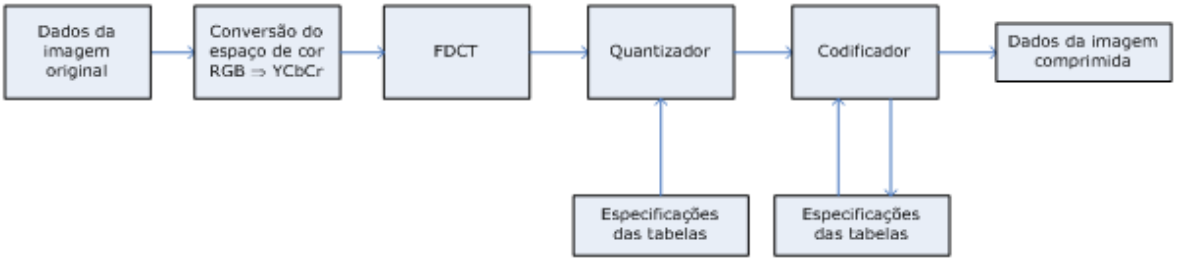

Figura 2.6: Processo de codificação mais detalhado

A norma JPEG permite a codificação de imagens com uma única componente, imagens em tons de cinza, e imagens com múltiplas componentes (com três planos de cor (RGB), por exemplo).

No caso de imagens RGB, os três planos de cor são convertidos noutra representação, YCbCr (o plano Y contém a informação de luminância, Cb e Cr a informação de cor), sendo

usual a sub-amostragem dos planos Cb e Cr, descrito com mais detalhe na secção 2.4. Este é o primeiro passo onde se introduz perdas na qualidade da imagem e, simultaneamente, ganhos de codificação.

O procedimento seguinte é depois aplicado a cada plano:

No processo de codificação os pixels da imagem original são agrupados em blocos de 8x8 pixels e cada bloco é convertido pela transformada discreta do cosseno directa (FDCT) num conjunto de 64 valores, os coeficientes da DCT. O primeiro desses 64 valores é o coeficiente DC (valor médio da intensidade do bloco) e os outros 63 são os coeficientes AC (componentes de frequência espacial do bloco).

Cada um destes 64 coeficientes é então dividido por uma constante e arredondado para o inteiro mais próximo. A constante depende do coeficiente (DC ou AC) e da componente (Y, Cb ou Cr). Esta constante é denominada por factor de qualidade, já que ao actuar sobre esse valor é possível obter um ficheiro com pouco tamanho e pouca qualidade ou vice-versa. Esta é a operação de quantização, a qual introduz a maior parte das perdas.

Depois da quantização o coeficiente DC e os 63 coeficientes AC são preparados para a codificação. O coeficiente DC do bloco anterior é usado como estimativa para o coeficiente DC actualmente quantizado e apenas a diferença é codificada (codificação diferencial ou DPCM). Os restantes 63 coeficientes AC são convertidos numa sequência linear utilizando um varrimento em Zig-Zag (para preservar a informação de frequência espacial) e automaticamente agrupar os possíveis zeros obtidos. Esta sequência linear é então codificada sem perdas, utilizando métodos estatísticos.

Nesta fase pode ser usado um de dois métodos de codificação de entropia, codificação de Huffman ou codificação aritmética, sendo a primeira a mais usual. Nesse caso, as tabelas de Huffman têm de ser conhecidas tanto pelo codificador como pelo descodificador (são calculadas para a imagem em questão ou usadas tabelas por defeito).

2.2 Organização do ficheiro codificado

Um ficheiro JPEG encontra-se dividido em diversos blocos de dados. Cada bloco é identificado por um marcador, um conjunto de bits da forma 0xFFaa, em que aa varia consoante o marcador [4].

Alguns dos marcadores mais importantes:

0xFFD8,	Start Of Image (SOI). Inicio da imagem.
0xFFEn,	APPn (n= 0..F). Reservado para uso de aplicações.
0xFFCn,	Start Of Frame n (SOFn).
	Inicio de frame, em que n indica o modo de compressão
	(n = 0, 1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15).
0xFFDB,	Quantization Table(s) (DQT). Tabela(s) de Quantização.
0xFFC4,	Huffman Table(s) (DHT). Tabela(s) de Huffman.
0xFFDD,	Define Restart Interval (DRI).
0xFFDm,	Restart interval m (RSTm), (m = 0..7).
0xFFDA,	Start Of Scan (SOS). Dados comprimidos.
0xFFD9,	End Of Image (EOI). Fim da imagem.

Os marcadores usados num ficheiro JPEG começam sempre com um ou mais 0xFF, seguido de um byte diferente de zero, que identifica a função do segmento de código seguinte. Para que não exista ambiguidade entre os marcadores e os dados codificados do ficheiro, sempre que exista um 0xFF na zona de dados codificados, segue-se sempre um byte a zero unicamente com esse propósito.

Existem duas categorias de marcadores, aqueles sem parâmetros e aqueles seguidos de um tamanho variável de parâmetros. Para o segundo caso, o primeiro parâmetro (2 bytes) é exactamente o tamanho da sequência de parâmetros, incluindo o parâmetro do tamanho e excluindo o marcador que define o segmento. A norma JPEG garante que fora dos segmentos os marcadores são únicos.

A estrutura de um ficheiro JPEG é a seguinte, sendo que não existe uma ordem predefinida [1]:

SOI
- DQT, tamanho, definições da(s) tabela(s) de quantização
- DRI, tamanho, intervalos de recomeço
  - SOFn, tamanho, parâmetros do frame
    - DHT, tamanho, definições da(s) tabela(s) de Huffman
    - SOS, tamanho, parâmetros
      - Dados comprimidos associados a RST0
      - ...
      - Dados comprimidos associados a RSTm
      - Dados comprimidos associados ao último RST
    - DHT, tamanho, definições da(s) tabela(s) de Huffman
    - SOS, tamanho, parâmetros
      - ...

EOI

2.3 Estrutura detalhada de um ficheiro JPEG

Marcador Início de Imagem (SOI)

- SOI (2 bytes)

Marcador Inicio de Frame (SOFn)

- SOFn (2 bytes)
- Lf (2 bytes) – Tamanho do segmento incluindo os 2 bytes do tamanho
- P (1 byte) – Precisão em bits (8 para Baseline JPEG)
- Y (2 bytes) – Número de linhas
- X (2 bytes) – Número de amostras/linha
- Nf (1 byte) – Número de componentes do frame (3 – cor; 1 – cinzentos)

- Para cada Nf (i=1, .., Nf)
  - C_i (1 byte) – Identificador de componente
  - H_i (4 bits mais significativos) – Factor de amostragem horizontal
  - V_i (4 bits menos significativos) – Factor de amostragem vertical
  - Tq_i (1 byte) – Número da tabela de quantização

Marcador Tabela Quantização (DQT)

- DQT (2 bytes)
- Lq (2 bytes) – Tamanho do segmento incluindo os 2 bytes do tamanho
- Para cada DQT (enquanto tamanho > 0)
  - Pq (4 bits mais significativos) – Precisão (0 – 8 bit; 1 – 16 bit)
  - Tq (4 bits menos significativos) – Identificador da tabela
  - Para (k=0, .., 63)
    - Q_k (1 ou 2 bytes) – Valores (armazenados no formato Zig-Zag)

Marcador Tabela Huffman (DHT)

- DHT (2 bytes)
- Lh (2 bytes) – Tamanho do segmento incluindo os 2 bytes do tamanho
- Para cada DHT (enquanto tamanho > 0)
  - Tc (4 bits mais significativos) – Classe da tabela
  - Th (4 bits menos significativos) – Identificador da tabela
    - Para (i=1, .., 16)
    - L_i (1 byte) – Número de códigos de tamanho i
- Para (i=1, .., 16)
- Para (j=1, .., 16)
  - V_{ij} (1 byte) – Valores associados a cada código de Huffman

Marcador Início de Dados (SOS)

- SOS (2 bytes)
- Ls (2 bytes) – Tamanho do segmento incluindo os 2 bytes do tamanho
- Ns (1 byte) – Número de componentes
- Para cada Ns (k=1, .., Ns)
  - Cs_k (1 byte) – Identificador de componente
  - Td_k (4 bits mais significativos) – Identificador de valores DC
  - Ta_k (4 bits menos significativos) – Identificador de Valores AC
- Ss (1 byte) – Início de selecção espectral
- Se (1 byte) – Fim de selecção espectral
- Ah (4 bits mais significativos) – Posição do bit de aproximação sucessiva
- Al (4 bits menos significativos) – Posição do bit de aproximação sucessiva

Marcador Fim de Imagem (EOI)

- EOI (2 bytes)

2.4 Compressão de imagens com múltiplas componentes

A norma JPEG permite codificar ficheiros com mais do que uma componente, por exemplo RGB. No caso de uma imagem em tons de cinza o ficheiro tem apenas uma componente (Y), enquanto que, no caso de uma imagem a cores, o ficheiro tem três componentes (Y, Cb, Cr).

No caso de imagens RGB, os três planos de cor são convertidos noutra representação, YCbCr (o plano Y contém a informação de luminância, Cb e Cr a informação de cor), sendo usual a sub-amostragem dos planos Cb e Cr.

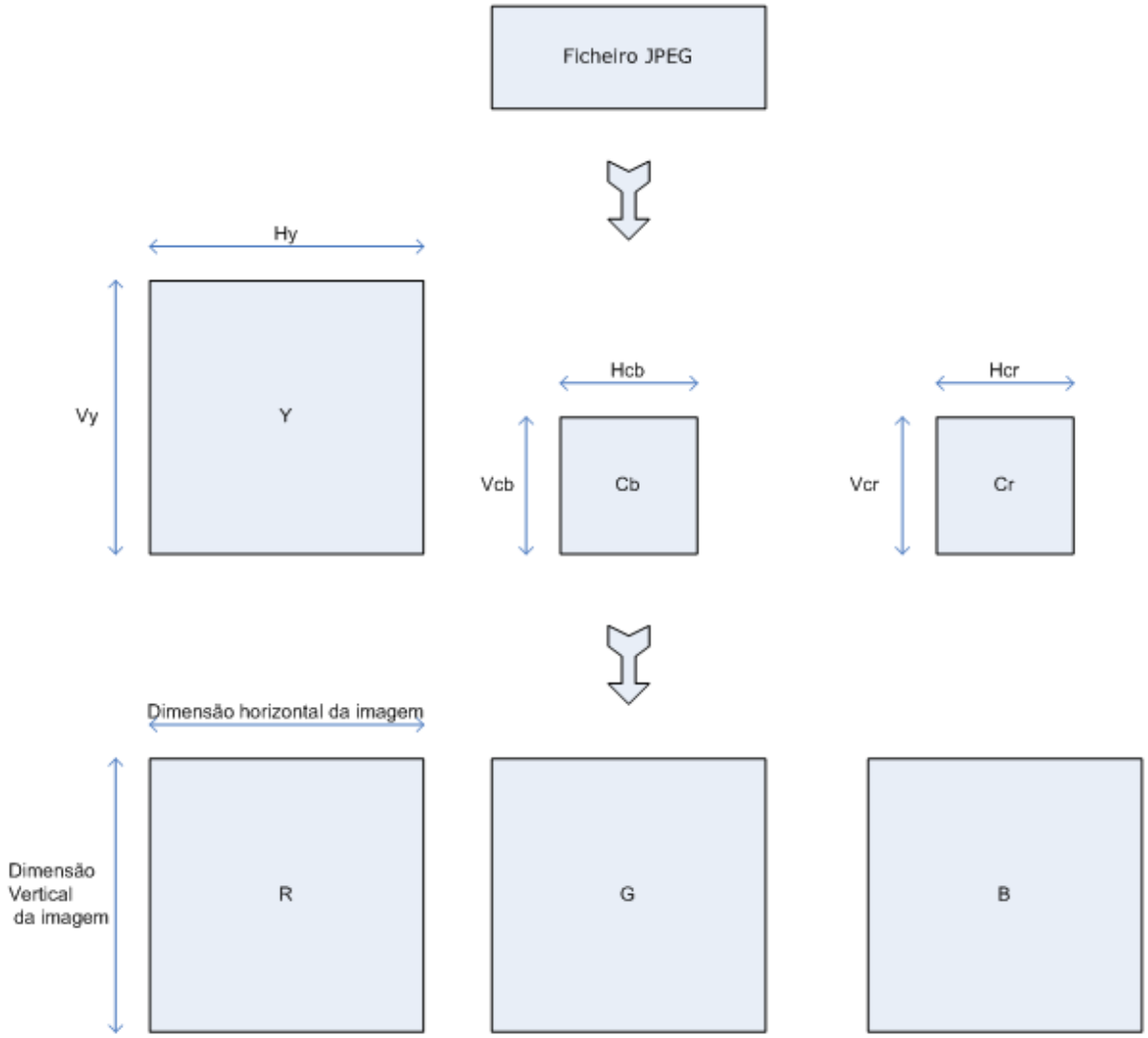

Figura 2.7: Compressão de imagens com múltiplas componentes

A conversão entre RGB e YCbCr é feita da seguinte forma:

$$R = Y + 1.402(Cr - 128) \tag{2.1a}$$

$$G = Y - 0.34414(Cb - 128) - 0.71414(Cr - 128) \tag{2.1b}$$

$$B = Y + 1.772(Cb - 128) \tag{2.1c}$$

$$Y = 0.299R + 0.587G + 0.114B \tag{2.2a}$$

$$Cb = -0.1687R - 0.3313G + 0.5B + 128 \tag{2.2b}$$

$$Cr = 0.5R - 0.4187G - 0.0813B + 128 \tag{2.2c}$$

Num ficheiro JPEG, a informação relativamente à cor encontra-se no formato YCbCr. Este formato é obtido através da conversão dos valores RGB utilizando as equações 2.1 e 2.2. No formato RGB temos as três componentes de cor (Red Green Blue) separadamente.

No formato YCbCr a componente Y contém a informação de luminância da imagem, ou seja, contém a informação sobre a intensidade de cor (ou luminosidade) da imagem. Os componentes Cb e Cr contêm a informação de cor da imagem, sendo que o componente Cb contém a informação relativa à cor azul (blue) e o componente Cr contém a informação relativa à cor vermelha (red).

O Cb e Cr normalmente são sub-amostrados para metade do seu tamanho original. Este primeiro passo, que permite algum ganho de codificação, aproveita uma característica do olho humano em que o mesmo é mais sensível a variações espaciais das intensidades luminosas do que das cores.

No cabeçalho JPEG temos acesso à informação acerca das dimensões de Y, Cb e Cr. Essas dimensões são relativas ao número de blocos 8x8 pixels que constitui cada componente. Por exemplo, se Hy = Vy = 2 e Hcb = Vcb = Hcr = Vcr = 1, então Y é constituído por quatro blocos 8x8 pixels (dois na horizontal e dois na vertical) e apenas um para Cb e outro para Cr. Este conjunto de blocos de Y, Cb e Cr tem o nome de Macrobloco. A dimensão da informação lida é sempre igual à dimensão de Y. Para este exemplo, a informação lida corresponde a 16x16 pixels conhecidos para as três componentes de cor RGB.

A norma JPEG prevê que as dimensões da imagem sejam múltiplas de 16 para se poder efectuar a descodificação/re-codificação do ficheiro sem problemas. Como isso raramente acontece, é necessário fazer uma expansão da imagem, tanto na horizontal como na vertical, de modo a que esta seja múltipla de 16, para tal são introduzidos zeros. Nesta altura pode-se descodificar/re-codificar a imagem sem problemas, sendo que a imagem resultante tem sempre as mesmas dimensões da imagem original, ou seja, a expansão referida anteriormente serve apenas para o efeito de descodificação/re-codificação.

Capítulo 3

Trabalho realizado

3.1 Descodificação

A primeira parte do trabalho incide sobre a descodificação de um ficheiro no formato JPEG. Uma visão global sobre o que se pretende pode ser representada pelas Figuras 3.1 e 3.2:

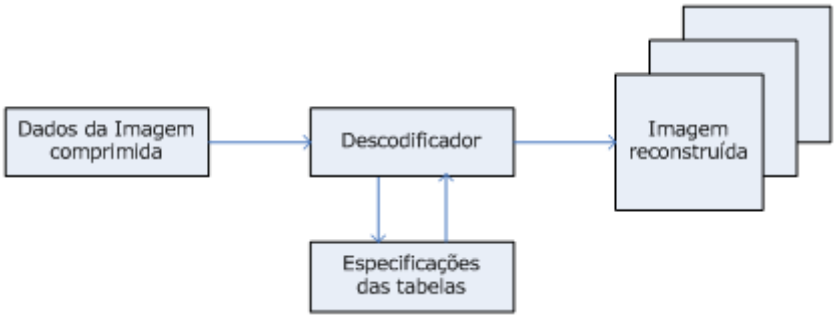

Figura 3.1: Descodificação

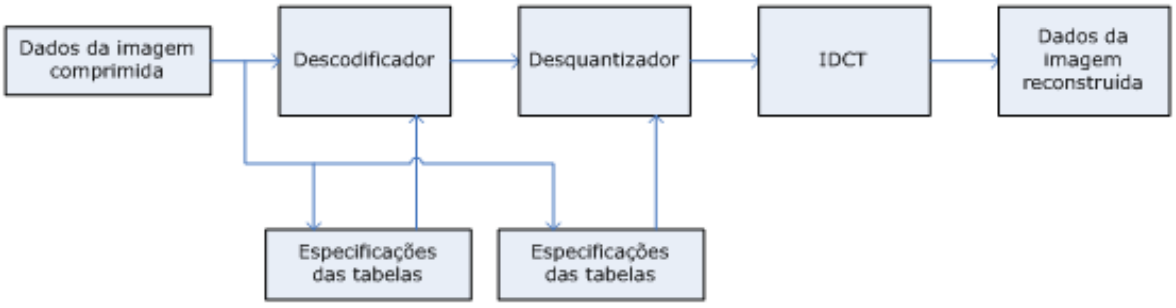

Figura 3.2: Processo de descodificação mais detalhado

Como o processo de compressão foi efectuado em blocos de 8x8 pixels, a descodificação tem que ser aplicada da mesma forma, isto é, é necessário descodificar individualmente blocos de 64 pixels e ir preenchendo a imagem descodificada.

A parte de descodificação com um pouco mais de detalhe pode ser representada através do diagrama da Figura 3.3:

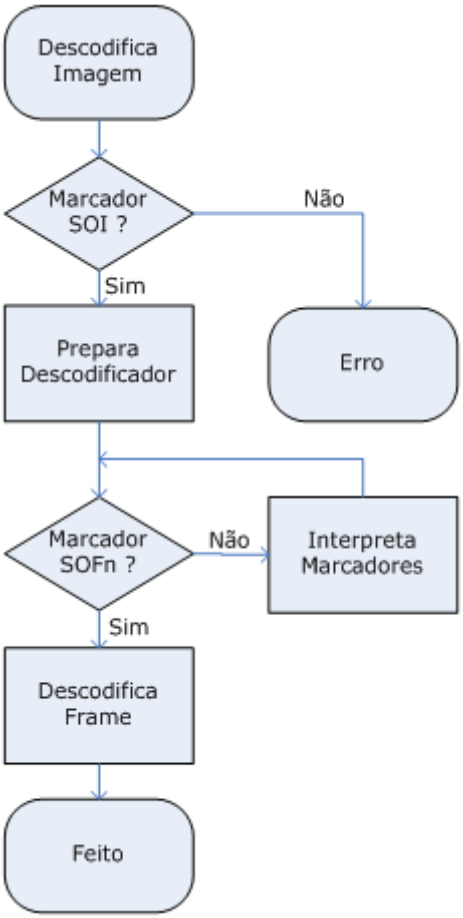

Figura 3.3: Diagrama da descodificação

3.1.1 Descodificação do cabeçalho

Para descodificar o ficheiro vamos precisar de conhecer dois tipos de tabelas que poderão ser fornecidas no ficheiro ou, na ausência destas, usar tabelas por defeito. Mais precisamente precisamos de ter a(s) DQT(s) (Tabela(s) de Quantização) e a(s) DHT(s) (Tabela(s) de Huffman).

Para ler a(s) DQT(s) é preciso ter em atenção a precisão dos valores (8 bits ou 16 bits). É necessário ler o identificador da mesma e de seguida podem ser lidos os valores tendo em atenção que estes são guardados de forma linear, tendo depois que ser convertidos para uma tabela (8x8) segundo a disposição da Figura 3.4 [3]:

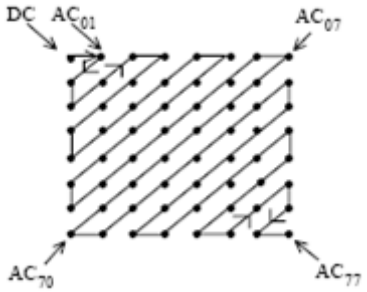

Figura 3.4: Formato Zig-Zag

O procedimento para construir as tabelas de Huffman envolve vários passos. Primeiro é preciso criar a tabela de tamanhos (Figura 3.5), segundo o seguinte fluxograma:

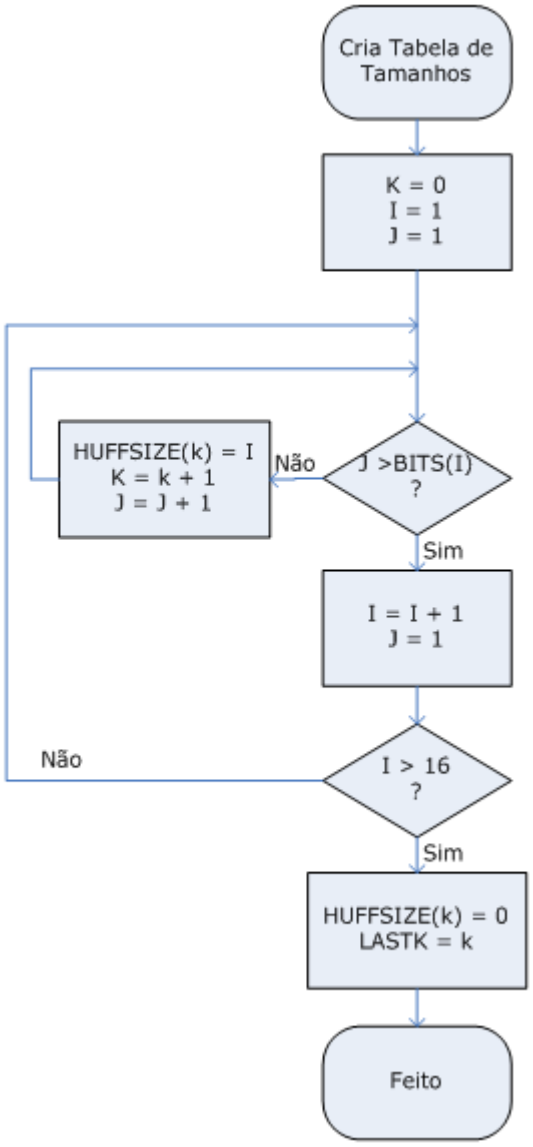

Figura 3.5: Tabela de tamanhos

Através deste procedimento vão ser lidos todos os códigos de Huffman e vai sendo guardada a informação relativamente aos tamanhos de cada um dos códigos. No fim deste procedimento são conhecidos quantos códigos existem para cada tamanho possível (de 1 a 16 bits) [2].

De seguida é necessário criar a tabela de códigos (Figura 3.6):

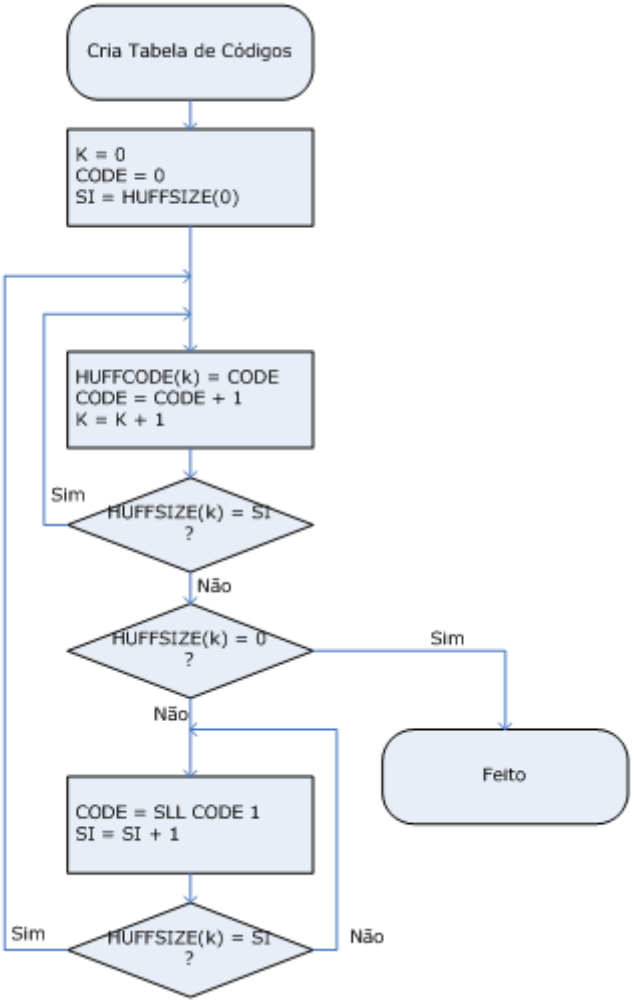

Figura 3.6: Tabela de códigos

Através deste procedimento os códigos de Huffman são todos lidos e guardados. Os códigos são guardados à medida que são lidos, sem qualquer ordem predefinida [2].

Por fim é necessário ordenar os códigos por tamanho (Figura 3.7):

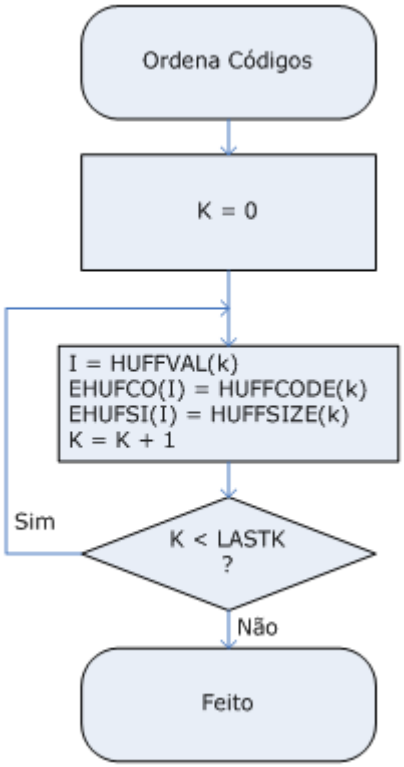

Figura 3.7: Procedimento para ordenar os códigos de Huffman por tamanho

Com este procedimento os códigos de Huffman ficam ordenados por tamanho.

EHUFCO contém todos os códigos de Huffman lidos, ordenados por tamanho de forma ascendente. EHUFSI contém os tamanhos de cada código de Huffman, segundo a mesma ordem [2].

Depois de preparadas as tabelas, é necessário ler a informação da zona de dados, bloco a bloco, e descodificar individualmente cada bloco de modo a termos o ficheiro JPEG descodificado.

3.1.2 Descodificação dos dados

Na compressão JPEG existem grandes sequências de zeros após a leitura em Zig-Zag (Figura 3.4) da matriz resultante da quantização dos coeficientes da DCT, por isso a codificação foi simplificada para apenas contar a ocorrência deste símbolo. Além disso, como apenas os zeros é que são contados, não é necessário que o símbolo apareça após o número de ocorrências. Então a codificação na compressão JPEG conta quantos zeros existem antes de um componente não zero e gera um par *(run, size)*, onde o campo Run indica o número de zeros que antecedem o valor não zero seguinte e o campo Size indica o número de bits necessários para ler o próximo símbolo.

A norma JPEG define em 15 o valor máximo do campo Run. Uma sequência de 16 ou mais zeros gera um par "Run/Size" especial, 15/0, que indica a existência de 15 zeros seguidos por mais um zero (16 zeros no total) e que reinicia o contador de zeros. Outro par especial é gerado quando a matriz de entrada termina com zeros, este par é 0/0 e indica que só existem zeros até ao fim da matriz.

Com toda a informação presente no cabeçalho JPEG correctamente processada e as tabelas conhecidas, chega a altura de descodificar a imagem. Nesta fase é necessário ler bit a bit a zona de dados até que seja encontrado um código de Huffman válido e feita a correspondência *código* $\Leftrightarrow$ *par (run, size)*. Por sua vez, *run* tem a informação sobre a quantidade de zeros que o bloco, de tamanho 8x8 pixels, tem seguidos até ao próximo valor diferente de zero. Este valor diferente de zero é obtido lendo o número de bits indicado por *size*.

Um diagrama deste procedimento pode ser visto na Figura 3.8:

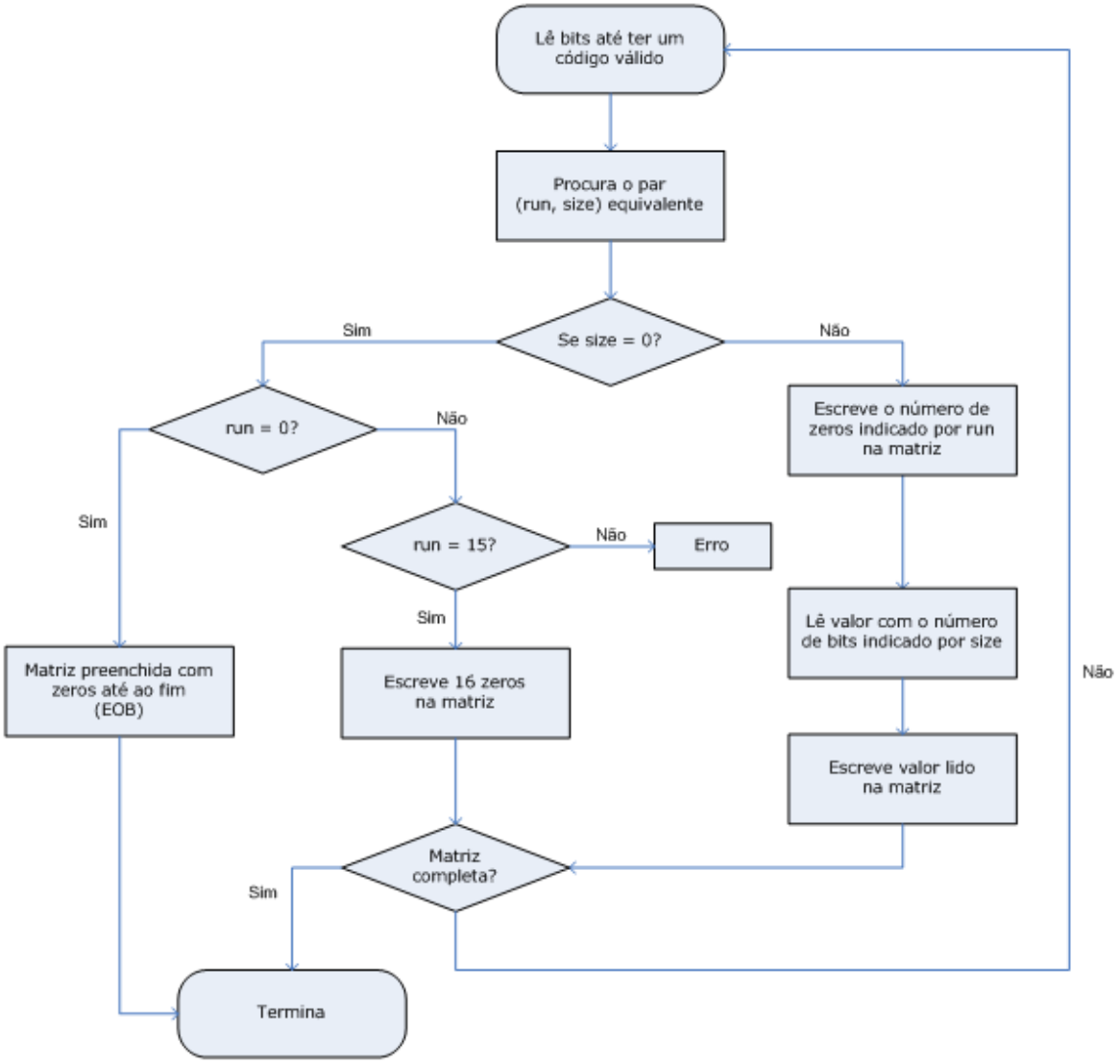

Figura 3.8: Descodificar imagem

O procedimento da Figura 3.8 tem algumas particularidades e é seguido de algumas operações que serão vistas de seguida.

Modo sequencial A descodificação de um bloco de 8x8 pixels é feita de modo sequencial. Os blocos são descodificados em sequência e um erro num bloco pode provocar erros em blocos seguintes ou até impossibilitar a correcta descodificação do ficheiro.

Tabelas É necessário ter em atenção a tabela de Huffman a usar. Para cada componente existe uma tabela de Huffman apropriada e apenas essa deve ser usada.

Mudança de escala Na fase da codificação os valores da amplitude a guardar são modificados. Na descodificação, antes de se usar o valor lido, é necessário fazer a operação inversa (uma alteração de escala) da seguinte forma:

size	**amplitude**
1	-1, 1
2	-3, -2, 2, 3
3	-7, . . . , -4, 4, . . . , 7
4	-15, . . . ,-8, 8, . . . , 15
5	-31, . . . , -16, 16, . . . , 31
6	-63, . . . , -32, 32, . . . , 63
7	-127, . . . , -64, 64, . . . , 127
8	-255,. . . , -128, 128,. . . , 255
9	-511, . . . , -256, 256, . . . , 511
10	-1023, . . . , -512, 512, . . . , 1023

Tabela 3.1: Mudança de escala

Por exemplo, para size = 2, temos o seguinte:

Original	**Alterado**
0	-3
1	-2
2	2
3	3

Tabela 3.2: Exemplo de mudança de escala

Codificação diferencial O primeiro valor a ser lido da zona de dados para cada bloco (para todos os componentes) é o coeficiente DC. Os 63 restantes são os coeficientes AC. O coeficiente DC é bastante importante porque indica o valor médio da intensidade dos pixels de todo o bloco.

Para poupar bits este valor é codificado de forma diferencial (ver exemplo na Figura 3.9), isto é, é codificada a diferença para o valor anterior sendo que para o primeiro valor da imagem (valor DC do bloco superior esquerdo), é feita a diferença com o valor zero. Isto

acontece sempre que não existam marcadores de reinicio (Restart intervals). Se existirem estes marcadores o DC é reinicializado a zero e é assegurado que o próximo código pertence ao primeiro de um bloco. Este assunto está explicado na secção 3.1.3.

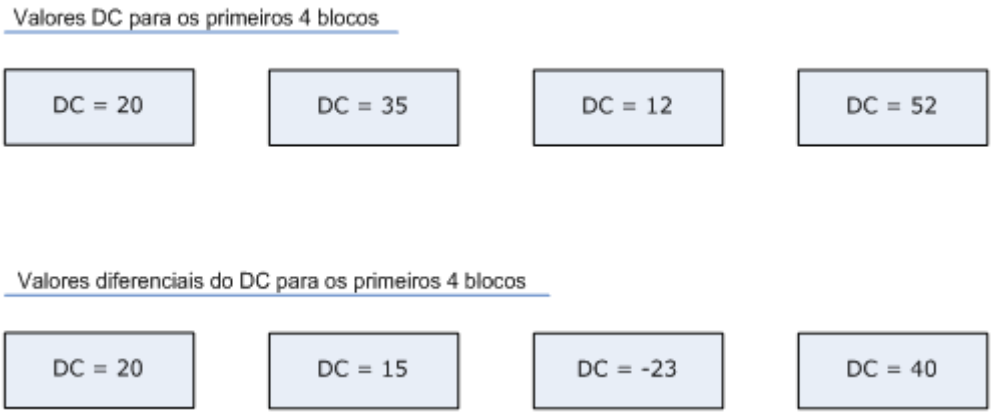

Figura 3.9: Codificação diferencial do DC

Algoritmo Nesta altura é executado o algoritmo da Figura 3.8 para 64 (DC + 63AC) valores de modo a completar um bloco de 8x8 pixels. De notar que para o DC o algoritmo é o mesmo da Figura 3.8, embora seja conhecido o valor de *run* que tem de ser obrigatoriamente zero.

Quantização Com o bloco de 8x8 pixels completo e depois do passo da Figura 3.9, é necessário multiplicar todos os valores do bloco pelos valores presentes na Tabela de Quantização. Neste passo é necessário, mais uma vez, ter em conta qual a tabela a ser usada em função do componente de cor (Y, Cb ou Cr).

Transformada discreta do cosseno inversa Depois da Tabela de Quantização é preciso aplicar a transformada discreta do cosseno inversa (IDCT). De seguida é necessário somar um offset (128) a todos os valores do bloco.

Nas tabelas seguintes estão representadas as três últimas operações que foram necessárias efectuar no acto da descodificação.

5	0	0	0	0	0	0	0
-3	0	0	0	0	0	0	0
1	-1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Tabela 3.3: Bloco depois de mudança de escala

-40	-2	0	1	0	-1	0	0
-19	0	-1	0	0	0	0	0
6	-7	0	1	0	0	0	0
0	0	0	0	-1	0	0	0
0	0	0	1	0	-1	0	1
0	0	0	0	-1	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0

Tabela 3.4: Bloco depois de Tabela de Quantização

129	129	130	130	131	132	132	132
130	130	130	131	131	131	131	131
131	131	131	131	131	130	130	130
133	133	132	132	131	131	130	130
134	134	133	133	132	132	131	131
135	135	135	134	134	134	134	134
135	136	136	136	136	137	137	137
136	136	136	137	138	138	139	139

Tabela 3.5: Bloco depois de IDCT

Arredondamento Finalmente é preciso garantir que, devido aos processos de arredondamentos envolvidos no acto de cálculo da IDCT, não hajam valores menores que zero (valor mínimo) nem superiores a 255 (valor máximo).

Macroblocos De seguida é efectuado o procedimento de descodificar um bloco para todos os blocos de um Macrobloco, de acordo com a Figura 2.7. Este procedimento é repetido para todos os Macroblocos do ficheiro.

Conversão de cor Quando a descodificação estiver concluída para todos os Macroblocos, estão preenchidas as 3 matrizes correspondentes às 3 componentes de cor (Y, Cb e Cr). Nesta altura apenas falta converter YCbCr para RGB. Esta operação é feita do seguinte modo, para cada pixel:

$$R = Y + 1.402(Cr - 128) \tag{3.1a}$$

$$G = Y - 0.34414(Cb - 128) - 0.71414(Cr - 128) \tag{3.1b}$$

$$B = Y + 1.772(Cb - 128) \tag{3.1c}$$

(Equação repetida da secção 2.4)

E assim ficam completas as três matrizes de cor RGB, todas do tamanho do ficheiro original. A cada pixel do ficheiro corresponde o valor na mesma posição das três componentes, como se pode ver no exemplo da Figura 3.10.

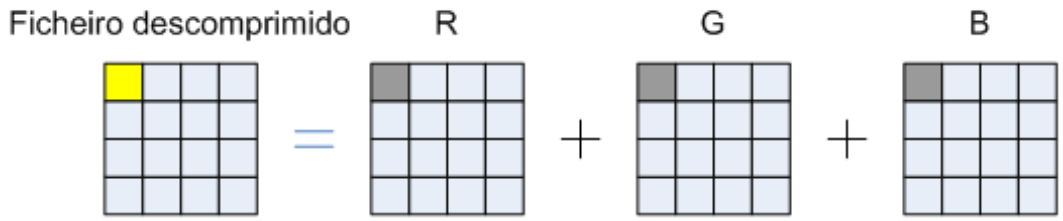

Figura 3.10: Ficheiro descomprimido

3.1.3 Marcas de controlo

Os marcadores de reinício (ou restart intervals) são os únicos marcadores que podem ser encontrados na zona de dados. Estes marcadores são bastante úteis para efeito de sincronismo em caso de erros no ficheiro. Como já foi visto anteriormente, um ficheiro JPEG está dividido em blocos que se agrupam em Macroblocos. Também foi visto que o primeiro coeficiente de cada bloco é o coeficiente DC que contém a informação sobre o valor médio das intensidades do mesmo, ou seja, é bastante importante para a correcta visualização de todos os valores do bloco. Existem 3 classes de coeficientes DC, um para cada componente de cor. A codificação diferencial (secção 3.2.3) é feita separadamente para cada um deles.

Assim sendo, um marcador de reinício indica que a próxima codificação diferencial é feita comparando o actual valor DC com zero (para todos os componentes). Indica também que se está perante o início de um Macrobloco.

O marcador de reinício tem de estar alinhado na zona de dados (Figura 3.11), de modo a que seja possível ser lido byte a byte ao invés de obrigar uma descodificação bit a bit. No caso da informação presente na zona de dados não estar alinhada antes de se colocar um marcador de reinício, é feito um padding de 1's até que isso aconteça, ou seja, o byte é acabado com bit's a 1 até que o alinhamento de verifique. Nesta altura já se pode colocar o marcador de reinício e prosseguir com a codificação.

O padding a 1's é possível de implementar graças aos códigos de Huffman utilizados, que garantem códigos unicamente descodificáveis, sendo que o início de um código nunca pode ser igual a outro código existente. Esta especificação garante que um padding de 1's não vai ser confundido com nenhum código. Depois do marcador encontrado, os bit's a 1 encontrados são descartados e começa uma nova leitura de um código.

Figura 3.11: Zona de dados

(a) - Códigos de Huffman

(b) - Últimos bits válidos (100)
Byte terminado com padding de 1's

(c) - Byte 0xFF (marcador). O byte seguinte tem de ser obrigatoriamente um marcador de reinício ou zero.

(d) - Byte 0xD0 (marcador de reinício).

Implica:

Descodificação de Macrobloco concluída

DC_Y = 0

DC_Cb = 0

DC_Cr = 0

Buffer de leitura reinicializado

Descodificação de novo Macrobloco

(e) - Códigos de Huffman relativos ao novo Macrobloco

3.2 Cópia e Codificação

A segunda parte do trabalho consistiu na re-codificação de alguns blocos, mantendo os outros inalterados.

Na altura de criar o ficheiro alterado é necessário alternar entre duas operações, copiar e codificar. Em primeiro lugar é necessário criar o ficheiro de destino e, byte a byte, copiar toda a zona do cabeçalho JPEG para o ficheiro de destino. Nesta operação é também copiada a informação do cabeçalho da zona de dados. O ficheiro de destino fica então pronto a receber a informação dos pixels codificados.

De seguida são percorridos todos os Macroblocos do ficheiro e, consoante tenha sido alterado ou não, é codificado ou copiado, respectivamente.

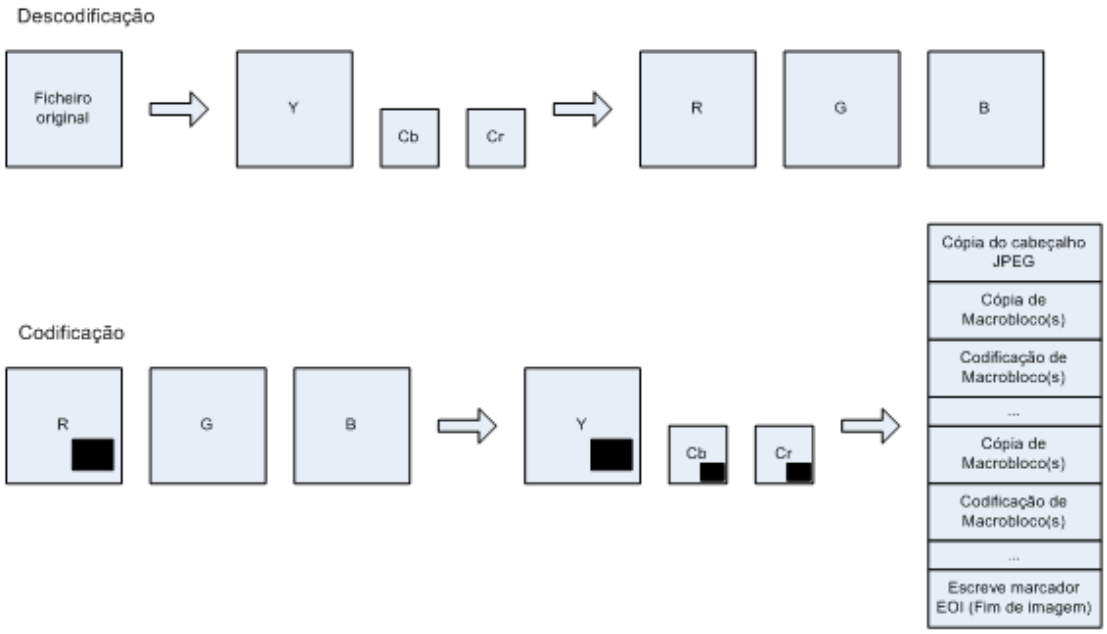

Figura 3.12: Descodificação e Codificação

3.2.1 Cópia

A operação de cópia está dividida em duas partes. Na primeira é copiada toda a informação relativa ao cabeçalho JPEG. Na segunda parte é feita uma cópia dos pixels codificados, Macrobloco a Macrobloco.

Cópia de cabeçalhos Nesta fase é feita uma cópia directa, byte a byte, do ficheiro de origem para o ficheiro de destino. Na fase de descodificação é guardado o valor do file pointer que corresponde ao início dos dados (já depois da informação de cabeçalho). Deste modo é feita a cópia até que seja atingido essa posição do file pointer e fica garantido que daí até final do ficheiro apenas ficará guardada a informação dos dados do ficheiro e, mesmo no final, o marcador EOI (End of image).

Cópia de Macroblocos Na cópia de Macroblocos são copiados todos os blocos correspondentes ao Macrobloco em questão. Nesta fase é lida bit a bit a zona de dados do ficheiro de origem e os bits lidos são escritos num buffer, partilhado com a função de codificação, para ser escrito no ficheiro de destino quando completar um byte.

É necessário ter um buffer de escrita partilhado com a função de codificação de modo que os dados escritos no ficheiro de destino sejam contínuos, já que vai existir alternância entre as funções de codificação e cópia de Macroblocos e os dados precisam estar seguidos, independentemente da função utilizada.

É também necessário ter especial atenção para o facto de sempre que for lido um byte 0xFF do ficheiro de origem, ser ignorado o byte seguinte (caso seja 0x00), que a única função é garantir que não se está perante nenhum marcador e não tem relevância nos dados. Assim sendo, na fase de escrita do buffer acima descrito, é necessário ter em conta que sempre que seja escrito no ficheiro de destino um byte a 0xFF é necessário escrever um byte com 0x00.

3.2.2 Codificação

A parte da codificação pode ser vista como o inverso da descodificação de um ficheiro no formato JPEG. Uma visão global sobre o assunto pode ser representada pelas Figuras 3.13 e 3.14:

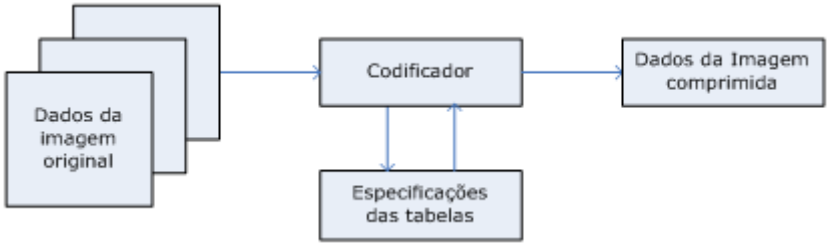

Figura 3.13: Codificação

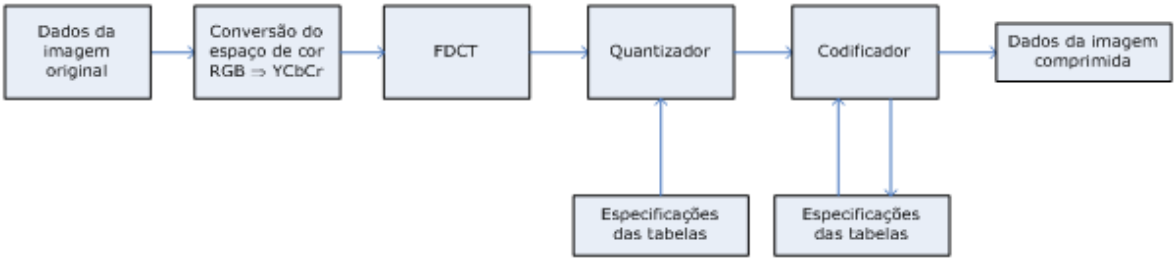

Figura 3.14: Processo de codificação mais detalhado

3.2.3 Princípio da codificação

Com toda a informação necessária para codificar presente nas matrizes de cor RGB, chega a altura de codificar a imagem. Tendo o respectivo bloco de 8x8 pixels pronto a codificar é necessário ler e codificar todos os 64 valores. Sempre que um valor lido seja diferente de zero é necessário verificar quantos zeros existem até ao próximo valor diferente de zero e, caso sejam menos de 16 zeros, pode ser codificada a correspondência *código* $\Leftrightarrow$ *par (run, size), amplitude*. *Run* é o número de zeros encontrados, *size* o número de bits necessários para ler *amplitude*, e *amplitude* é o próximo valor da matriz diferente de zero. Caso existam apenas zeros até ao fim da matriz é codificado o par (0, 0) indicado EOB (fim de bloco). Caso sejam encontrados 16 zeros seguidos e não estejamos na condição anterior é codificado o par (15, 0).

Um diagrama deste procedimento pode ser visto na Figura 3.15:

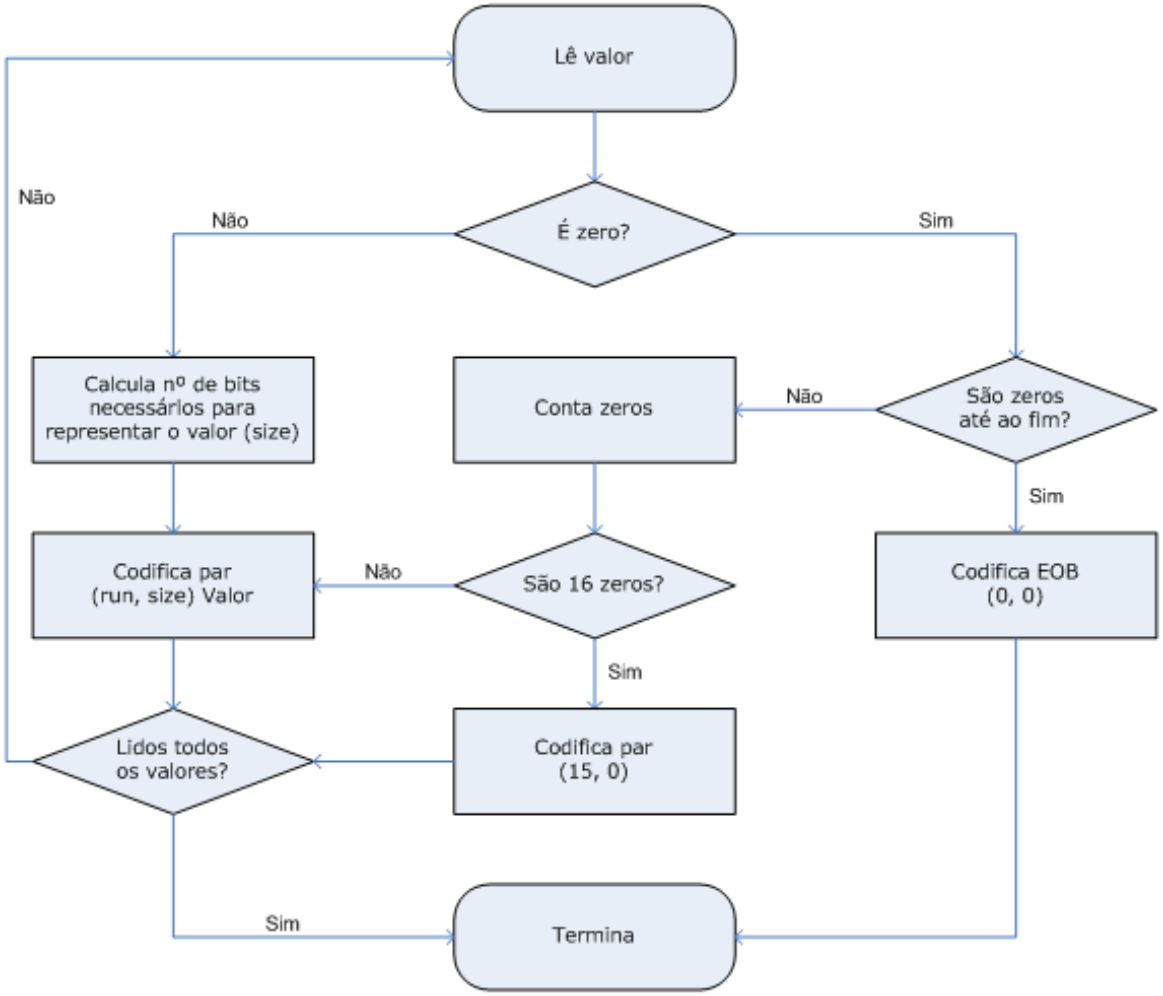

Figura 3.15: Codificar imagem

O procedimento da Figura 3.15 tem alguns pormenores e é antecedido de algumas operações que serão vistas de seguida.

Modo sequencial A codificação de um bloco de 8x8 pixels é feita de modo sequencial. Os blocos são codificados "uns atrás dos outros" e um erro num bloco pode provocar erros em blocos seguintes ou até provocar a impossibilidade de uma futura descodificação.

Tabelas É necessário ter em atenção a tabela de Huffman a usar. Para cada componente existe uma tabela de Huffman apropriada e apenas essa deve ser usada.

Conversão de cor A primeira operação a fazer aquando da codificação é converter RGB para YCbCr. Esta operação é feita do seguinte modo, para cada pixel, tendo em conta o procedimento de sub-amostragem dos componentes Cb e Cr em relação a Y, caso exista.

$$Y = 0.299R + 0.587G + 0.114B \tag{3.2a}$$

$$Cb = -0.1687R - 0.3313G + 0.5B + 128 \tag{3.2b}$$

$$Cr = 0.5R - 0.4187G - 0.0813B + 128 \tag{3.2c}$$

(Equação repetida da secção 2.4)

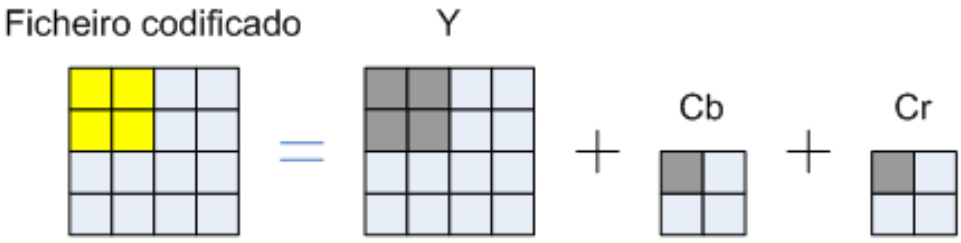

Figura 3.16: Ficheiro codificado

Macroblocos De seguida é efectuado o procedimento de codificar um bloco para todos os blocos de um Macrobloco. Este procedimento é repetido para todos os Macroblocos que tenham sido alterados.

Transformada discreta do cosseno directa Tendo a matriz de 8x8 pixels correspondente a um bloco a codificar, a primeira operação a fazer é subtrair um offset (128) a todos os valores do bloco. Depois é preciso aplicar a transformada discreta do cosseno directa (FDCT).

Quantização É depois necessário dividir, arredondando para o inteiro mais próximo, todos os valores do bloco pelos valores presentes na Tabela de Quantização. Neste passo é necessário, mais uma vez, ter em conta qual a tabela a ser usada em função do componente de cor (Y, Cb ou Cr).

Nas tabelas seguintes estão representadas as duas últimas operações que foram necessárias efectuar no acto da codificação.

129	129	130	130	131	132	132	132
130	130	130	131	131	131	131	131
131	131	131	131	131	130	130	130
133	133	132	132	131	131	130	130
134	134	133	133	132	132	131	131
135	135	135	134	134	134	134	134
135	136	136	136	136	137	137	137
136	136	136	137	138	138	139	139

Tabela 3.6: Bloco a codificar

-40	-2	0	1	0	-1	0	0
-19	0	-1	0	0	0	0	0
6	-7	0	1	0	0	0	0
0	0	0	0	-1	0	0	0
0	0	0	1	0	-1	0	1
0	0	0	0	-1	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0

Tabela 3.7: Bloco depois de FDCT

5	0	0	0	0	0	0	0
-3	0	0	0	0	0	0	0
1	-1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Tabela 3.8: Bloco depois de Tabela de Quantização

Codificação diferencial O primeiro valor a ser lido da matriz para cada bloco (para todos os componentes) é o coeficiente DC. Os 63 restantes são os coeficientes AC. À semelhança do que acontecia na descodificação, também na codificação apenas é codificada a diferença do valor DC do bloco corrente com o bloco anterior (da mesma componente).

Algoritmo Nesta altura é executado o algoritmo da Figura 3.15 para 64 (DC + 63AC) valores de modo a codificar um bloco de 8x8 pixels. De notar que para o DC o algoritmo é o mesmo da Figura 3.15, embora o valor de *run* tenha de ser obrigatoriamente zero.

Mudança de escala Antes de se usar o valor lido no ponto anterior, é necessário fazer uma alteração de escala de acordo com a Tabela 3.1, mas de forma inversa.

Escrever bits Finalmente os bits são escritos no ficheiro de destino como no exemplo da secção 3.2.4.

3.2.4 Exemplo de codificação de um bloco

De seguida estão representados os passos da codificação, desde o bloco 8x8 pixels a codificar até aos bits daí resultantes:

129	129	130	130	131	132	132	132
130	130	130	131	131	131	131	131
131	131	131	131	131	130	130	130
133	133	132	132	131	131	130	130
134	134	133	133	132	132	131	131
135	135	135	134	134	134	134	134
135	136	136	136	136	137	137	137
136	136	136	137	138	138	139	139

Tabela 3.9: Bloco a codificar

1	1	2	2	3	4	4	4
2	2	2	3	3	3	3	3
3	3	3	3	3	2	2	2
5	5	4	4	3	3	2	2
6	6	5	5	4	4	3	3
7	7	7	6	6	6	6	6
7	8	8	8	8	9	9	9
8	8	8	9	10	10	11	11

Tabela 3.10: Bloco depois de subtraído 128

-40	-2	0	1	0	-1	0	0
-19	0	-1	0	0	0	0	0
6	-7	0	1	0	0	0	0
0	0	0	0	-1	0	0	0
0	0	0	1	0	-1	0	1
0	0	0	0	-1	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0

Tabela 3.11: Bloco depois da FDCT

5	0	0	0	0	0	0	0
-3	0	0	0	0	0	0	0
1	-1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Tabela 3.12: Bloco depois de DQT

Run	Size	Amplitude
0	3	5
1	2	-3
0	1	1
4	1	-1
0	0	

Tabela 3.13: Run, Size e Amplitude

Código	Amplitude*
101	11
11011	0
00	1
111011	0
010	

Tabela 3.14: Código e Amplitude

* Valor com mudança de escala e codificação diferencial (DC)

Stream de bits
101111101100001111010010

Tabela 3.15: Stream de bits

3.3 Algoritmos para a Remoção do efeito de olhos vermelhos

A terceira parte do trabalho consiste na remoção do efeito dos olhos vermelhos. Para tal é necessário, em primeiro lugar, detectar a zona a corrigir, e de seguida aplicar-lhe um dos possíveis algoritmos.

3.3.1 Detecção da zona afectada

Para a detecção da zona afectada foi usado o seguinte conceito. A zona afectada tem uma grande incidência da cor vermelha e, simultaneamente, pouca incidência das cores verde e azul. Foi então desenvolvida uma fórmula que maximizasse essa relação de modo a ser facilmente detectável um zona com essas características (Equação 3.3):

$$K = \frac{red^2}{red^2 + green^2 + blue^2} \tag{3.3}$$

A equação 3.3 devolve um valor (entre 0 e 1) que mostra a incidência do vermelho em relação às outras cores. Assim, quanto maior for K maior é a incidência do vermelho sobre as outras cores.

Na Figura 3.17 pode ser visto um histograma das cores RGB num olho com o efeito de olho vermelho.

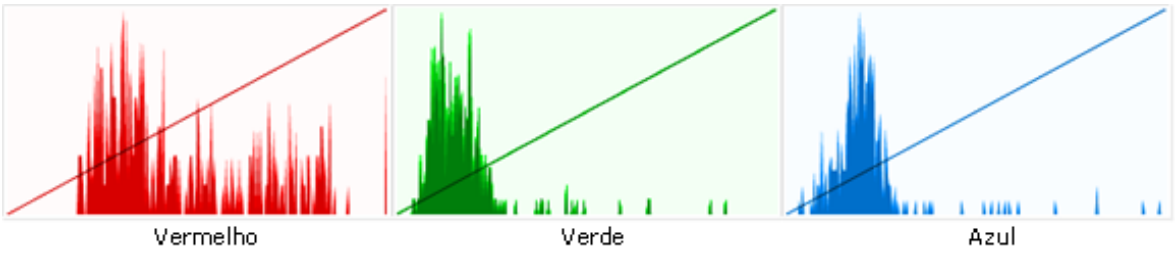

Figura 3.17: Histograma de cores

No histograma acima pode-se verificar que a maior parte de ocorrências das cores verde e azul estão nas baixas intensidades. As ocorrências da cor vermelha estão repartidas por toda a gama de intensidades. É neste aspecto que é necessário intervir, removendo as ocorrências da cor vermelha de média e alta intensidade.

3.3.2 Algoritmos

Foi implementado um algoritmo existente (Algoritmo 2) e desenvolvidos três novos algoritmos (Algoritmo 1, 3 e 4) de remoção dos olhos vermelhos de modo a verificar qual obtinha melhores resultados.

3.3.3 Algoritmo 1

O primeiro algoritmo funciona segundo o seguinte princípio: sabendo que o pixel está na zona afectada (secção 3.3.1), os valores dos pixels R, G e B são alterados segundo a equação 3.4:

$$\begin{aligned} red &= red * 0.5 \\ green &= green * 1.3 \\ blue &= blue * 1.2 \end{aligned} \tag{3.4}$$

Os valores escolhidos para este algoritmo têm como objectivo reduzir a intensidade da cor vermelha e aumentar a intensidade das cores verde e azul nas mesmas proporções, de modo a tentar manter constante a média de cores e consequente valor DC.

A Figura 3.18 mostra uma fotografia com o efeito de olhos vermelhos e a Figura 3.19 é a mesma fotografia depois de aplicado este algoritmo. Na Figura 3.20 temos a diferença das duas anteriores.

Figura 3.18: Imagem 1

Figura 3.19: Algoritmo 1

Figura 3.20: Diferença para o algoritmo 1

3.3.4 Algoritmo 2

O segundo algoritmo ([17]) funciona segundo o seguinte princípio: sabendo que o pixel está na zona afectada (secção 3.3.1), os valores dos pixels R, G e B são alterados segundo a equação 3.5:

$$\begin{aligned} red &= red * 0.513 \\ green &= green * 1.0 \\ blue &= blue * 0.193 \end{aligned} \tag{3.5}$$

A Figura 3.21 mostra uma fotografia com o efeito de olhos vermelhos e a Figura 3.22 é a mesma fotografia depois de aplicado este algoritmo. Na Figura 3.23 temos a diferença das duas anteriores.

Figura 3.21: Imagem 2

Figura 3.22: Algoritmo 2

Figura 3.23: Diferença para o algoritmo 2

3.3.5 Algoritmo 3

O terceiro algoritmo funciona segundo o seguinte princípio: sabendo que o pixel está na zona afectada (secção 3.3.1), os valores dos pixels R, G e B são alterados segundo a equação 3.6:

$$red = \text{red} * (1 - \text{K}) \tag{3.6}$$

Este algoritmo pretende reduzir a intensidade da cor vermelha de uma forma proporcional para todas as intensidades, de acordo com a incidência da mesma (K, secção 3.3.1). Ao diminuir a intensidade da cor vermelha, está-se simultaneamente a remover o efeito de olhos vermelhos enquanto se preserva a cor natural do olho, permitindo assim obter um resultado mais natural.

A Figura 3.24 mostra uma fotografia com o efeito de olhos vermelhos e a Figura 3.25 é a mesma fotografia depois de aplicado este algoritmo. Na Figura 3.26 temos a diferença das duas anteriores.

Figura 3.24: Imagem 3

Figura 3.25: Algoritmo 3

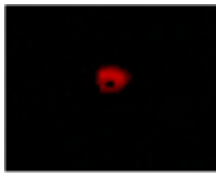

Figura 3.26: Diferença para o algoritmo 3

3.3.6 Algoritmo 4

O quarto algoritmo funciona segundo o seguinte princípio: sabendo que o pixel está na zona afectada (secção 3.3.1), os valores dos pixels R, G e B são alterados segundo a equação 3.7:

$$red = \frac{\text{red} * 255}{\sqrt{255^2 + red^3}} \tag{3.7}$$

Este algoritmo pretende reduzir a intensidade da cor vermelha de uma forma menos acentuada nas baixas intensidades e de uma forma mais acentuada nas intensidades mais altas, de acordo com a incidência da mesma (K, secção 3.3.1). Ao diminuir a intensidade da cor vermelha, está-se simultaneamente a remover o efeito de olhos vermelhos enquanto se preserva a cor natural do olho, permitindo assim ter um resultado mais natural.

A Figura 3.27 mostra uma fotografia com o efeito de olhos vermelhos e a Figura 3.28 é a mesma fotografia depois de aplicado este algoritmo. Na Figura 3.29 temos a diferença das duas anteriores.

Figura 3.27: Imagem 4

Figura 3.28: Algoritmo 4

Figura 3.29: Diferença para o algoritmo 4

Capítulo 4

Resultados

Imagem		
Nome	Resolução (Pixels)	Tamanho (Bytes)
Img1.jpg	1704 x 2272	1.367.071
Img2.jpg	3072 x 2304	2.934.022
Img3.jpg	3072 x 2304	2.965.634
Img4.jpg	3072 x 2304	3.174.012

Tabela 4.1: Dados da imagem

GIMP (Qualidade 85%)		GIMP (Qualidade 100%)		Proposto	
Tamanho (Bytes)	PSNR (dB)	Tamanho (Bytes)	PSNR (dB)	Tamanho (Bytes)	PSNR (dB)
489.198	41.0	1.820.889	50.0	1.367.464	60.7
843.782	39.5	3.900.372	46.8	2.934.151	63.6
876.159	38.9	3.923.088	44.0	2.965.761	72.2
1.010.363	38.4	4.121.994	43.8	3.174.555	67.9

Tabela 4.2: Resultados

As imagens usadas nas experiências tentaram representar as várias situações possíveis do dia a dia, isto é, as imagens foram obtidas variando a distância ao indivíduo, tamanho e intensidade do efeito de olhos vermelhos.

Quando as imagens são processadas por outra aplicação (GIMP [6]) verifica-se que para um factor de qualidade de 85% o tamanho do ficheiro baixa em relação ao original e a sua qualidade é inferior à da aplicação proposta.

Usando a mesma aplicação (GIMP) mas usando um factor de qualidade de 100% verifica-se que o tamanho do ficheiro aumenta significativamente e mesmo assim apresenta uma qualidade inferior à da aplicação proposta.

A aplicação proposta mantém o tamanho do ficheiro bastante idêntico ao original (ligeiramente superior) mas apresenta uma qualidade sempre significativamente superior ao de outras aplicações.

Nas figuras seguintes pode-se ver quais os pixels modificados (cor branca), e consequente perda de qualidade da imagem, após remover o efeito de olhos vermelhos.

Na Figura 4.1 foi usada a aplicação GIMP com um factor de qualidade de 85% e verifica-se que a maior parte dos pixels foram alterados.

Na Figura 4.2 foi usada a aplicação GIMP com um factor de qualidade de 100% verificando-se que mesmo assim muitos dos pixels sofreram alterações.

Finalmente na Figura 4.3 foi usada a aplicação proposta e facilmente se verifica que apenas os pixels dos blocos onde está presente o indesejado efeito foram modificados. Todos os outros se mantiveram exactamente iguais, sendo esse o principal objectivo deste trabalho.

Figura 4.1: GIMP 85%

Figura 4.2: GIMP 100%

Figura 4.3: Aplicação proposta

Alguns exemplos de imagens com o efeito de olhos vermelhos antes de depois de serem processadas por este software:

Figura 4.4: Imagem 1 original

Figura 4.5: Imagem 1 corrigida

Linha de comando usada:

./redEyeRemoval -b 76 109 87 119 -b 132 112 143 122 red_effect_1.jpg

Figura 4.6: Imagem 2 original

Figura 4.7: Imagem 2 corrigida

Linha de comando usada:

./redEyeRemoval -b 43 56 58 69 -b 134 46 149 61 red_effect_2.jpg

Como se pode verificar através desta fotografia, o algoritmo de remoção do efeito de olhos vermelhos usado por defeito (Algoritmo 4, secção 3.3.6) mostra-se bastante eficaz na remoção da tonalidade avermelhada do olho. Ao mesmo tempo, este algoritmo é capaz de reconhecer a cor original do olho, ficando assim os olhos com a cor original, obtendo-se um resultado final mais realista.

Como se pode verificar, os reflexos que os olhos com o efeito de olhos vermelhos continham permaneceram inalterados, mesmo depois da fotografia ter sido processada por este software.

Estas são algumas das vantagens deste algoritmo desenvolvido de raiz.

Figura 4.8: Imagem 3 original

Figura 4.9: Imagem 3 corrigida

Linha de comando usada:

./redEyeRemoval -b 98 157 117 173 -b 168 155 183 171 -k 40 red_effect_3.jpg

Figura 4.10: Imagem 4 original

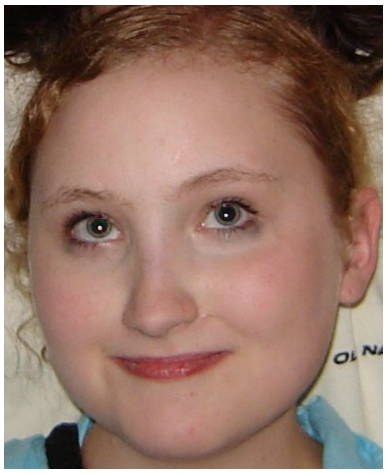

Figura 4.11: Imagem 4 corrigida

Linha de comando usada:

./redEyeRemoval -b 93 233 121 260 -b 234 220 265 248 -k 75 -a 4 red_effect_4.jpg

Figura 4.12: Imagem 5 original

Figura 4.13: Imagem 5 corrigida

Linha de comando usada:

./redEyeRemoval -b 684 647 713 675 -b 873 644 900 670 -b 145 703 160 718 -b 215 696 231 713 -k 50 red_effect_5.jpg

Figura 4.14: Imagem 6 original

Figura 4.15: Imagem 6 corrigida

Linha de comando usada:

./redEyeRemoval -b 1442 770 1495 827 -b 1785 874 1834 931 red_effect_6.jpg

Figura 4.16: Imagem 7 original

Figura 4.17: Imagem 7 corrigida

Linha de comando usada:

./redEyeRemoval -b 46 28 63 48 red_effect_7.jpg

Capítulo 5

Conclusões

O trabalho desenvolvido focou, numa primeira fase, o estudo da norma JPEG e o desenvolvimento de um descodificador. Deste trabalho resultou uma biblioteca com um conjunto de funções para manipulação de ficheiros de imagem em formato JPEG e um programa que, com o auxílio da biblioteca desenvolvida, efectua o processamento dos diversos blocos de um ficheiro JPEG.

Numa segunda fase foi desenvolvido um codificador que apenas re-codifica os blocos alterados e copia todos os outros de modo a que não exista perda de qualidade em toda a imagem.

Numa última fase foram desenvolvidos alguns algoritmos de remoção do efeito de olhos vermelhos e foram efectuados alguns testes de modo a comparar os resultados obtidos através dos mesmos.

Grande parte das aplicações "comerciais" para processamento de imagens disponibiliza "plugins" para remoção do efeito de olhos vermelhos. O grande problema é que todas essas aplicações re-codificam toda a imagem e, como o JPEG é um formato de compressão com perdas, isso vai degradar a qualidade da imagem e eventualmente aumentar o tamanho do ficheiro resultante.

A principal contribuição deste trabalho foi o desenvolvimento de uma aplicação para remoção do efeito de olhos vermelhos em fotografia digital no formato JPEG que apenas re-codifica os Macroblocos alterados. A biblioteca para manipulação de imagem em formato JPEG é também importante, não só para este trabalho mas também para possíveis trabalhos futuros.

Do ponto de vista de resultados obtidos verifica-se que as imagens processadas pela ferramenta desenvolvida ficam com muito melhor qualidade, quer do ponto de vista objectivo (PSNR) quer subjectivo (visualmente) do que quando processadas com outra aplicação (GIMP). Usando a mesma aplicação (GIMP) e usando um factor de qualidade de 100% verifica-se que o tamanho do ficheiro aumenta significativamente e mesmo assim apresenta uma qualidade inferior à da aplicação proposta. A aplicação proposta mantém o tamanho do ficheiro bastante idêntico ao original (ligeiramente superior) mas apresenta uma qualidade sempre significativamente superior ao de outras aplicações.

Capítulo 6

Anexo

6.1 Manual do software

Do trabalho realizado resultou o desenvolvimento de uma biblioteca para manipulação de ficheiros no formato JPEG e no desenvolvimento de duas ferramentas:

- Um descodificador JPEG (converte JPEG para PPM (*Portable Pixelmap*))

- Uma ferramenta para remoção do efeito dos olhos vermelhos

O objectivo da biblioteca é a de facultar um conjunto de funções e estruturas de dados que permita a manipulação de ficheiros em formato JPEG.

A biblioteca está dividida em alguns módulos:

- "encode" – Envolve todo o processo de codificação do ficheiro

- "decode" – Envolve todo o processo de descodificação do ficheiro

- "bitio" – Envolve todo o processo de manipulação de bits

- "leedct" – Envolve os cálculos relativos à DCT

- "common" – Envolve partes comuns a todos os módulos

Para utilizar as ferramentas desenvolvidas utilizam-se os seguintes comandos:

jpeg-decoder

jpeg-decoder [Nome do ficheiro]

[Nome do ficheiro] ⇒ O nome do ficheiro deve incluir a extensão (jpg ou jpeg)

Como resultado é gerado 1 ficheiro:

decodedImage.ppm ⇒ Imagem descodificada

redEyeRemoval

redEyeRemoval [-h] [-v] [-a n] [-k c] [-b PHi PVi PHf PVf] [-b ...]
[Nome do ficheiro.jpg]

redEyeRemoval ⇒ Nome do executável

-h ⇒ Mostra a ajuda

-v ⇒ Verbose mode. Aumenta o nível de informação exibida na consola

-a ⇒ Indica que vai ser passado como parâmetro o algoritmo de remoção a usar
n ⇒ Número do algoritmo a usar: [1..4]

-k ⇒ Constante para detecção do efeito de olho vermelho
c ⇒ Valor a usar: [0..100]

-b ⇒ Indica que vão ser passadas como parâmetros as coordenadas (pixeis) da área afectada.
Pode ser repetido para 100 areas diferentes.
PHi ⇒ Valor horizontal do pixel superior esquerdo da zona afectada
PVi ⇒ Valor vertical do pixel superior esquerdo da zona afectada
PHf ⇒ Valor horizontal do pixel inferior direito da zona afectada
PVf ⇒ Valor vertical do pixel inferior direito da zona afectada

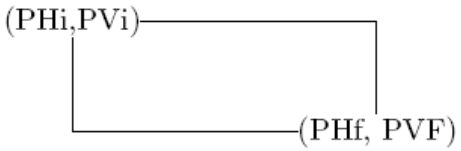

Figura 6.1: Área afectada

[Nome do ficheiro] ⇒ O nome do ficheiro deve incluir a extensão (jpg ou jpeg)

Exemplo:

redEyeRemoval -v -a 4 -k 85 -b 2 2 50 50 -b 200 2 250 50 filename.jpg

Como resultado são gerados 3 ficheiros:

decodedImage.ppm ⇒ Imagem descodificada

redEyeRemoved.ppm ⇒ Imagem descodificada com remoção do efeito

redEyeRemoved.jpg ⇒ Imagem re-codificada com remoção do efeito

Este software foi concebido para ser versátil no seu modo de funcionamento.

Ao correr este software sem qualquer argumento, o próprio software é capaz de pesquisar na fotografia por áreas avermelhadas usando valores predefinidos e, caso não encontre nenhuma área, processa a imagem de destino como uma cópia da original.

Se encontrar alguma área avermelhada, processa a fotografia a partir desse ponto até ao final, removendo essa e todas as áreas avermelhadas encontradas.

Caso seja passada como argumento uma área a remover o efeito de olhos vermelhos, o software apenas processa a fotografia nessa área, não alterando qualquer dado noutra área da mesma.

Se forem passadas várias áreas para ser removido o efeito de olhos vermelhos (até um máximo de 100) apenas essas áreas são processadas, ficando todo o resto da fotografia inalterada em relação ao original.

Se alguma, ou todas, das áreas passadas como argumentos para remoção do efeito de olhos vermelhos não tiverem de facto uma zona avermelhada, ou a tonalidade da cor vermelha seja inferior ao da constante utilizada por defeito pelo software (esta constante (K) pode ser passada como parâmetro), o software é capaz de reconhecer esse facto e, pura e simplesmente, ignora essa área não efectuando qualquer alteração nos dados da mesma. Este facto não impossibilita que o efeito de olhos vermelhos seja removido noutra das áreas introduzidas, porque o software trata cada e todas as áreas a remover o efeito de forma individual.

Capítulo 7

Referências

[1] Pennebacker, William B., Mitchell Joan L., *"JPEG still image data compression standard"*, placeStateNew York, 1993

[2] ISO/IEC, *"Information Technology – Digital Compression And Coding Of Continuous-Tone Still Images – Requirements And Guidelines"*, ITU Recommendation T.81, 1992

[3] Wallace, Gregory k., *"The JPEG Still Picture Compression Standard"*, IEEE Transactions on Consumer Electronics, 1991

[4] CityplaceHamilton, Eric, *"JPEG File Interchange Format"*, Version 1.02, 1992

[5] http://www.cs.sfu.ca/CC/365/li/material/notes/Chap4/Chap4.2/Chap4.2.html

[6] http://www.gimp.org/

[7] http://www.jpeg.org/jpeg/index.html

[8] http://cnx.org/content/m11096/latest/

[9] http://stargate.ecn.purdue.edu/~ips/tutorials/jpeg/jpegtut1.html

[10] http://www.w3.org/Graphics/JPEG/

[11] http://en.wikipedia.org/wiki/JPEG

[12] D. Salomon, *"Data compression - The complete reference"*, Springer, 2000, 2nd edition

[13] http://www.videnet.gatech.edu/cookbook.pt/list_page.php?topic=3&url =mjpeg.htm&level=1&sequence=4&name=Motion%20JPEG%20(MJPEG)

[14] Honthaner, Eve Light, *"The Complete Film Production Handbook"*, 2001, Third Edition

[15] Miano, John, *"Compressed Image File Formats: JPEG, PNG, GIF, XBM, BMP'*, 1999, Pap/Cdr Edition

[16] Taubman, David, Marcellin, Michael, *"JPEG2000: Image Compression Fundamentals, Standards and Practice"*, Springer, 2001, 1st Edition

[17] http://registry.gimp.org/plugin?id=4212